UNIVERSIDAD NACIONAL DE LA PLATA

Programacion Concurrente - Resolucion Examenes

Tomatis, Vicens, Torchia, Olmos, Manzin, Petraccaro, Traberg, Villareal

(Parcial MC - 2020 - 1 - Tema 6)

Consigna:
Resolver con PASAJE DE MENSAJES ASINCRÓNICOS (PMA) el siguiente problema. Se debe simular la atención en un banco con 3 cajas para atender a N clientes que pueden ser especiales (son las embarazadas y los ancianos) o regulares. Cuando el cliente llega al banco se dirige a la caja con menos personas esperando y se queda ahí hasta que lo terminan de atender y le dan el comprobante de pago. Las cajas atienden a las personas que van a ella de acuerdo al orden de llegada pero dando prioridad a los clientes especiales; cuando terminan de atender a un cliente le debe entregar un comprobante de pago. Nota: maximizar la concurrencia. **Respuesta:**

Proceso Persona

```
chan coordinador
chan clienteEspecial[0..2]
chan clienteRegular[0..2]
chan empleado[0..P-1]
chan respuesta[0..P-1]
chan termine
chan sync
chan syncCaja[0..2]
boolean especial

Process Persona[0..P-1]
    send coordinador(id)
    send sync
    recv respuesta[id](idE)
    if especial{
        send clienteEspecial[idE](id)
    }else{
        send clienteRegular[idE](id)
    }
    send syncCaja[idE]()
    recv empleado[id](comprobante)
    send termine(idE)
    send sync
```

Proceso Coordinador

```
Process Coordinador
array cajas[0..2] [(0), 2]
    while True{
        recv sync
        if !empty(termine){
            recv termine(idE)
            cajas(idE)--
        }else{
            recv coordinador(idC)
            idE = min(cajas)
            cajas(idE)++
            send respuesta[idC](idE)
        }
    }
```

Proceso Caja

```
Process Empleado[0..2]
    while True{
        recv syncCaja[id]()
        if !empty(clienteEspecial[id]){
            recv clienteEspecial[id](idC)
        }else{
            recv clienteRegular[id](idC)
        }
        comprobante = generarComprobante()
        send empleado[idC](comprobate)
    }
```

(Parcial MC - 2020 - 1 - Tema 6)

Consigna:
Resolver con PMS (Pasaje de Mensajes SINCRONICOS) el

siguiente problema. En una exposición aeronáutica hay un simulador de vuelo (que debe ser usado con exclusión mutua) y un empleado encargado de administrar el uso del mismo. A su vez hay P personas que van a la exposición y solicitan usar el simulador, cada una de ellas espera a que el empleado lo deje acceder, lo usa por un rato y se retira para que el empleado deje pasar a otra persona. El empleado deja usar el simulador a las personas respetando el orden en que hicieron la solicitud. Nota: cada persona usa sólo una vez el simulador. **Respuesta:**

Proceso Persona

```
1
2  Process Persona[id:0..P-1]{
3      Empleado!llegue(id)
4      Empleado?usar()
5      //usa el simulador
6      Empleado!sali()
7  }
```

Proceso Empleado

```
1  Process Empleado{
2      bool libre=true
3      do
4      [] Persona[*]?llegue(id) =>
5          if libre{
6              Persona[id]!usar()
7              libre=false
8          }else{
9              push(fila, id)
10         }
11     [] Persona[*]?sali() => {
12         if empty(fila){
13             libre=true
14         }else{
15             pop(fila, id)
16             Persona[id]!usar()
17         }
18     }
19     od
20 }
```

(Parcial MC - 2020 - 1 - Tema 6)

Consigna:
Resolver con PMS (Pasaje de Mensajes SINCRÓNICOS) el siguiente problema. Simular la atención de una estación de servicio con un único surtidor que tiene un empleado que atiende a los N clientes de acuerdo al orden de llegada. Cada cliente espera hasta que el empleado lo atienda y le indica qué y cuánto cargar; espera hasta que termina de cargarle combustible y se retira. Nota: cada cliente carga combustible sólo una vez; todos los procesos deben terminar. **Respuesta:**

Proceso Cliente

```
Process Cliente[id:0..P-1]{
    text carga
    BufferPedido!pedido(carga, id)
    Empleado?Termine()
    -- Se va
}
```

Proceso Buffer

```
Process BufferPedido{
    text carga
    int idC
    cola [text, int] pedidos
    for i=1 to N*2{
        do
        [] Cliente[*]?pedido([carga,idC]) =>
    push(pedidos,[carga,idC])
        [] !empty(pedidos);Empleado?Listo() =>
    Empleado!Siguiente(pop(pedidos))
    }
}
```

Proceso Empleado

```
Process Empleado{
    text carga
    int idC
    for i=1 to P{
        BufferPedido!Listo()
        BufferPedido?Siguiente([carga,idC])
        -- Le carga Nafta
        Cliente[idC]!Termine()
    }
}
```

(Parcial MC - 2020 - 1 - Tema 6)

Consigna:
En una carrera hay C corredores y 3 Coordinadores. Al llegar los corredores deben dirigirse a los coordinadores para que cualquiera de ellos le dé el número de "chaleco" con el que van a correr y luego se va. Los coordinadores atienden a los corredores de acuerdo al orden de llegada (cuando un coordinador está libre atiende al primer corredor que está esperando). Nota: maximizar la concurrencia. **Respuesta:**

Proceso Corredor

```
Process Corredor[id:0..C-1]{
    int nro
    BufferPedido!llegue(id)
    BufferNumero!avisar(id)
    BufferNumero?miNumero(nro)
    -- Se va
}
```

Proceso Empleado

```
Process Coordinador[0..2]{
    int idC, nro
    while True{
        BufferPedido!Listo(id)
        BufferPedido?Siguiente(idC)
        nro = generarNumero(idC)
        BufferNumero!camiseta(nro)
    }
}
```

Proceso BufferPedido

```
Process BufferPedido{
    int idC, idCo
    cola int pedidos
    for i=1 to C*2{
        do
        [] Corredor[*]?llegue(idC) => push(pedidos,idC)
        [] !empty(pedidos);Coordinador[*]?listo(idCo)
    => Coordinador[idCo]!Siguiente(pop(pedidos))
    }
}
```

Proceso BufferNumero

```
Process BufferNumero{
    int idC, nro
    cola int camisetas
    for i=1 to C*2{
        do
        [] Coordinador[*]?camiseta(nro) =>
    push(camisetas, nro)
        [] !empty(camisetas);Corredor[*]?listo(idC) =>
    Corredor[idC]!miNumero(pop(camisetas))
    }
}
```

()

Consigna:

3.Resolver el siguiente problema con ADA. Hay un sitio web para identificación genética que resuelve pedidos de N clientes . Cada cliente trabaja continuamente de la siguiente manera: genera la secuencia de ADN, la envía al sitio web para evaluar y espera el resultado; después de esto puede comenzar a generar la siguiente secuencia de ADN. Para resolver estos pedidos el sitio web cuenta con 5 servidores idénticos que atienden los pedidos de acuerdo al orden de llegada (cada pedido es atendido por un único servidor). Nota: maximizar la concurrencia. Suponga que los servidores tienen una función ResolverAnálisis que recibe la secuencia de ADN y devuelve un entero con el resultado.

Respuesta:

Proceso Cliente

```
procedure ejercicio1
    task type Cliente
        entry repuesta(resultado: IN text);
    end cliente;

    arrClientes=array 1..N of Cliente

    task body Cliente is
    int adn
    begin
        loop
            adn=generarADN()
            BufferADN.mandarADN(adn,id)
            accept repuesta(resultado IN text) do
                ver(resultado)
            end respuesta
        end loop;
    end;
begin
end.
```

Proceso BufferADN

```
    task type BufferADN
        entry mandarADN(adn, id: IN int);
        entry listo(idCli, adnOUT: OUT int);
    end BufferAND;

    task body BufferADN is
    begin
        loop
            accept listo(OUT idCli, OUT adnOUT) do
                accept mandarADN(adnIN, idCliente IN
↪   int) do
                    idcli=idCliente
                    adnOUT=adnIN
                end mandarADN
            end listo
        end loop
```

Proceso Servidor

```
    task type Servidor
    end Servidor

    arrServidores = array 1..N of Servidores

    task body Servidor is
    text res
    begin
        loop
            BufferADN.listo(idCli,adn)
            res=ResolverAnálisis(adn)
            arrClientes[idCli].repuesta(res)
        end loop
```

()

Consigna:
2- Resolver con ADA la siguiente situación. En una obra social que tiene 15 sedes en diferentes lugares se tiene información de las enfermedades de cada uno de sus clientes (cada sede tiene sus propios datos). Se tiene una Central donde se hacen estadísticas, y para esto repetidamente elige una enfermedad y debe calcular la cantidad total de clientes que la han tenido. Esta información se la debe pedir a cada Sede. Maximizar la concurrencia. Nota: existe una función ElegirEnfermos(e) que es llamada por cada Sede y devuelve la cantidad de clientes de esa sede que han tenido la enfermedad e.

Respuesta:

Proceso Sede

```
task type Sede is
end sede;
sedes arr[1..15] of Sede

task body Sede is
  text enfermedad
  int cant
  loop
    Central.informarEnfermedad(enfermedad)
    Central.obtenerCantidad(ElegirEnfermos(enfermedad))
  end loop;

end Sede

```

Proceso Central

```
task Central is
    entry informarEnfermedad(out text enfermedad);
    entry obtenerCantidad(in text enfermedad);
end Central;

task body Central is
    total:Integer
begin
    loop
        enfer=ElegirEnfermedad()
        total=0
        for 1..15*2 loop
            select
                accept informarEnfermedad(out text enfermedad)
                ↪ do
                    enfermedad=enfer
                end informarEnfermedad;
            or
                accept obtenerCantidad(in int cantidad) do
                    total+=cantidad
                end obtenerCantidad;
            end select;
        end loop;
        Put_Line("la cantidad es "+Integer'Image(total))
    end loop;
end Central

```

()

Consigna:
En una playa hay 5 equipos de 4 personas cada uno (en total son 20 personas donde cada una conoce previamente a que equipo pertenece). Cuando las personas van llegando esperan con los de su equipo hasta que el mismo esté completo (hayan llegado los 4 integrantes), a partir de ese momento el equipo comienza a jugar. El juego consiste en que cada integrante del grupo junta 15 monedas de a una en una playa (las monedas pueden ser de 1, 2 o 5 pesos) y se suman los montos de las 60 monedas conseguidas en el grupo. Al finalizar cada persona debe conocer el grupo que más dinero junto. Nota: maximizar la concurrencia. Suponga que para simular la búsqueda de una moneda por parte de una persona existe una función Moneda() que retorna el valor de la moneda encontrada. **Respuesta:**

Proceso Persona

```
Procedure ejercicio x

task type Persona is
    entry empezar()
    entry resultado(ganadores: IN Integer)
end Persona;

Personas = array[1..20] of Persona

task body Persona is
    int misMonedas
begin
    accept recibirId(id)
    Grupos[miGrupo].llegue(id)
    accept empezar()
    for i=1 to 15
    begin
        misMonedas+=moneda()
    end for
    Grupos[miGrupo].informar(misMonedas, miGrupo)
    accept resultado(ganadores: IN Integer)
end Persona
```

Proceso Grupo

```
task type Grupo is
    entry llegue(idPersona: IN Integer)
    entry informar(monedas, id: IN Integer)
end Grupo

Grupos = array[1..5] of Grupo

task body Grupo is
    cola int integrantes
    int monedasGrupo
begin
for i=1 to 4
begin
    accept llegue(idPersona)
    push(integrantes,idPersona)
end for
for i=1 to 4
    Personas[pop(integrantes)].empezar()
end for
for i=1 to 4
    accept informar(monedas, id : IN Integer)
    monedasGrupo+=monedas
end for
```

Proceso Coordinador

```
task type Coordinador is
    entry subtotal(monedas, Grupo: IN Integer)
end Coordinador;

task body Coordinador is
    total = array[1..5] of int
    int ganadores
begin
for i=1 to 5
begin
    accept subtotal(monedas, Grupo: IN Integer)
    total[Grupo]+=monedas
end for
    ganadores=max(total)
for i=i to 20
begin
    Persona.resultado(ganadores)
end for
end Coordinador

begin
    for i=1 to 20
    begin
        Persona.recibirId(i)
    end for
end Procedure x
```

()

Consigna:
En un sistema para acreditar carreras universitarias, hay UN Servidor que atiende pedidos de U Usuarios de a uno a la vez y de acuerdo con el orden en que se hacen los pedidos. Cada usuario trabaja en el documento a presentar, y luego lo envía al servidor; espera la respuesta de este que le indica si está todo bien o hay algún error. Mientras haya algún error,vuelve a trabajar con el documento y a enviarlo al servidor. Cuando el servidor le responde que está todo bien, el usuario se retira. Cuando un usuario envía un pedido espera a lo sumo 2 minutos a que sea recibido por el servidor, pasado ese tiempo espera un minuto y vuelve a intentarlo (usando el mismo documento). **Respuesta:**

Proceso Usuarios

```
Procedure Y

task type Usuario is
end Usuario

Usuarios = array[1..U] of Usuario

task body Usuario is
begin
    trabajo=Laburo() -- (no como cristian)
    while errores loop
    begin
        select
            Servidor.mandarLaburo(trabajo,errores)
        or delay(120)
            delay(60)
        end select
        if errores
            trabajo=Laburo() -- (no como cristian)
        end if
    end loop
```

Proceso Servidor

```
task Servidor is
    entry mandarLaburo(trabajo: IN text, errores: OUT boolean)
end Servidor

task body Servidor
begin
    loop
        accept mandarLaburo(trabajo: IN text, errores: OUT
        ↪  boolean) do
            errores=corregir(trabajo)
        end mandarLaburo
    end loop
end
```

()

Consigna:
Resolver con ADA el siguiente problema. Simular la venta de entradas a un evento musical por medio de un portal web. Hay N clientes que intentan comprar una entrada para el evento; los clientes pueden ser regulares o especiales (clientes que están asociados al sponsor del evento). Cada cliente especial hace un pedido al portal y espera hasta ser atendido; cada cliente regular hace un pedido y si no es atendido antes de los 5 minutos, vuelve a hacer el pedido siguiendo el mismo patrón (espera a lo sumo 5 minutos y si no lo vuelve a intentar) hasta ser atendido. Después de ser atendido, si consiguió comprar la entrada, debe imprimir el comprobante de la compra.

El portal tiene Z entradas para vender y atiende los pedidos de acuerdo al orden de llegada pero dando prioridad a los Clientes Especiales. Cuando atiende un pedido, si aún quedan entradas disponibles le vende una al cliente que hizo el pedido y le entrega el comprobante.

Nota: no debe modelarse la parte de la impresión del comprobante, sólo llamar a una función Imprimir (comprobante) en el cliente que simulará esa parte; la cantidad E de entradas es mucho menor que la cantidad de clientes (E « C); todas las tareas deben terminar.

Respuesta:

Proceso ClienteRegular

```
Procedure E

task type ClienteRegular is
end ClienteRegular

ClientesRegulares = array[1..R] of ClienteRegular

task body ClienteRegular is
begin
    while !atendido loop
        select
            Portal.pedirRegular(entrada, comprobante)
            atendido=true
        or delay(5)
            null
        end select
    end loop
    if entrada!=-1
        imprimir(comprobante)
    end if
end
```

Proceso ClienteEspecial

```
task type ClienteEspecial is
end ClienteEspecial

ClientesEspeciales = array[1..CE] of ClienteEspecial

task body ClienteEspecial is
begin
    while !atendido loop
        Portal.pedirEspecial(entrada)
        atendido=true
    end loop
end
```

Proceso Portal

```
task Portal is
    entry pedirRegular(entrada : OUT integer, comprobate : OUT
    ↪ text)
    entry pedirEspecial(entrada : OUT integer)
    entry
end Portal

task body Portal is
    int vendidas
begin
    for i=1 to N
        select
            accept pedirEspecial(entrada : OUT integer)
                entrada=-1
                if entradasEntregadas<E
                    entrada=i
                    vendidas++
                end if
            end pedirEspecial
        or
            when (pedirEspecial'count=0) =>
            accept pedirRegular(entrada : OUT  integer,
            ↪ comprobate : OUT text)
                entrada=-1
                comprobante = null
                if entradasEntregadas<E
                    entrada=i
                    comprobante=generarComprobate(i)
                    vendidas++
                end if
            end pedirRegular
        end select
end
```

Proceso Portal

```
task Portal is
    entry pedirRegular(entrada : OUT integer)
    entry
end Portal

task body Portal is
begin
    int cantvendidas=0
    for i=1 to N loop
        entradaAsig=-1
        if cantvendidas <= E
            entradaAsig=i
        end if
        select
            accept pedirEspecial(entrada : OUT integer) do
                   entrada=entradaAsig
            end pedirespecial
        or when pedirEspecial'count==0 =>
            accept pedirRegular(entrada : OUT  integer,
            ↪ comprobate : OUT text)
                entrada=entradaAsig
                comprobante=generarComprobate(i)
            end pedirEspecial
        end select
        if cantvendidas <= E
            cantvendidas++
        end if
    end loop
end
```

Consigna:
Resolver con ADA el siguiente problema. Simular la atención en una oficina donde atiende un empleado. Hay N personas que deben hacer UN trámite cada uno. Cuando una persona llega le da los datos al empleado y espera a que este resuelva el trámite y le entregue un comprobante. Por otro lado está el director de la oficina, el cual necesita 5 informes del empleado; cada vez que necesita un informe se lo pide al empleado y si no lo atiende inmediatamente sigue trabajando durante 10 minutos y luego lo vuelve a intentar siguiendo el mismo patrón (si no lo atiende inmediatamente trabaja durante 10 minutos y luego lo vuelve a intentar) hasta que el empleado lo atienda y le dé el informe pedido. El empleado atiende los pedidos de acuerdo al orden de llegada pero siempre dando prioridad a los pedidos de las personas. Nota:
todas las tareas deben terminar. **Respuesta:**

Proceso Persona

```
Procedure ejercicio Oficina

task type Persona is
end Persona;

Personas = array[1..N] of Persona

task body Persona is
    text datos, respuesta
begin
    Empleado.enviarDatos(datos,respuesta)
end Persona
```

Proceso Empleado

```
task type Empleado is
    entry enviarDatos(datos : IN integer, respuesta :
    ↪ OUT text)
    entry pedirInformes(informe : OUT text)
end Empleado

task body Empleado is
    text informe
begin
for i=1 to N+5
begin
    select
        accept enviarDatos(datos : IN integer, respuesta
        ↪ : OUT text)
            respuesta=analizar(datos)
        end enviarDatos
    or
        when (enviarDatos'count=0) =>
            accept pedirInformes(informe : OUT text)
                informe=generarInforme()
            end pedirInformes
    end select
end for
end
```

Proceso Director

```
task type Director is
end Director

task body Director is
    cola text informes
    int informesRecibidos
    text informe
begin
    while informesRecibidos<5 loop
        select
            empleado.pedirInformes(informe)
            push(informes, informe)
            informesRecibidos++
        or
            delay(10) --minutos
        end select
    end loop
end
```

Consigna:
Resolver con ADA el siguiente problema. Un programador contrató a 5 estudiantes para testear los sistemas desarrollados por él. Cada estudiante tiene que trabajar con un sistema diferente y debe encontrar 10 errores (suponga que seguro existen 10 errores en cada sistema) que pueden ser: urgentes, importantes, secundarios. Cada vez que encuentra un error se lo reporta al programador y espera hasta que lo resuelva para continuar. El programador atiende los reportes de acuerdo al orden de llegada pero teniendo en cuenta las siguientes prioridades: primero los urgentes, luego los importantes y por último los secundarios; si en un cierto momento no hay reportes para atender durante 5 minutos trabaja en un nuevo sistema que está desarrollando. Después de resolver los 50 reportes de error (10 por cada estudiante) el programador termina su ejecución. Nota: todas las tareas deben terminar. **Respuesta:**

Proceso Estudiante

```
Procedure ejercicio Laboratorio

task type Estudiante is
end Estudiante;

Estudiantes = array[1..5] of Estudiante

task body Estudiante is
    text datos, respuesta
begin
    for i=1 to 10
        error=encontrarError()
        if error="urgente"
        begin
            programador.urgente(error)
        elsif error="importante"
            programador.importante(error)
        else
            progrmador.secundario(error)
        end if
end Persona
```

Proceso Programador

```
task type Programador is
    entry urgente(error: IN text)
    entry importante(error: IN text)
    entry secundario(error: IN text)
end Programador

task body Programador is
begin
    while corregidos<50 loop
        select
            accept urgente(error)
                debbugear(error)
            end urgente
            corregidos++
        or when (urgente'count=0) =>
            accept importante(error)
                debbugear(error)
            end importante
            corregidos++
        or when (urgente'count=0) and
        ↪ (importante'count=0) =>
            accept secundario(error)
                debbugear(error)
            end secundario
            corregidos++
        else
            delay(5) --minutos
        end select
    end loop
end
```

Consigna:

2) Resolver con PMA el siguiente problema. Para una aplicación de venta de pasaje se tienen 3 servidores replicados para mejorar la eficiencia en la atención. Existen N clientes que hacen alguna de estas dos solicitudes: compra de un pasaje o devolución de un pasaje. Las solicitudes se deben atender dando prioridad a las solicitudes de compra. Nota: suponga que cada cliente llama a la función TipoSolicitud() que le devuelve el tipo de solicitud a realizar. Maximizar la concurrencia. **Respuesta:**

Proceso Cliente

```
chan text, int comprarBuffer
chan text, int devolucionesBuffer
chan int listo
chan sync
chan text, int tarea[0..2]
chan int respuesta[1..N]
Process Cliente[1..N]{
    text solicitud
    int res
    solicitud = TipoSolicitud()
    if solicitud="comprar"{
        send compraBuffer(solicitud,id)
    }else{
        send devolucionBuffer(solicitud,id)
    }
    send sync
    recv respuesta[id](res)
}
```

Proceso Buffer

```
Process Buffer{
    int idServer, idCliente
    text solicitud
    while True{
        recv listo(idServer)
        recv sync
        if empty(comprarBuffer){
            recv devolucionesBuffer(solicitud,idCliente)
        }else{
            recv compraBuffer(solicitud,idCliente)
        }
        send tarea[idServer](solicitud,idCliente)
    }
}
```

Proceso Servidor

```
Process Servidor[0..2]{
    text solicitud
    int idCliente, ticket, dinero
    while True{
        send listo(id)
        recv tarea[id](solicitud,idCliente)
        if solicitud="compra"{
            ticket = generarTicket()
            send respuesta[idCliente](ticket)
        }else{
            dinero = devolverDinero(idCliente)
            send respuesta[idCliente](dinero)
        }
    }

}
```

Consigna:
Resolver con ADA el siguiente problema. Se debe simular un juego en el que participan 30 jugadores que forman 5 grupos de 6 personas. Al llegar cada jugador debe buscar las instrucciones y el grupo al que pertenece en un cofre de cemento privado para cada uno; para esto deben usar un único martillo gigante de a uno a la vez y de acuerdo al orden de llegada. Luego se debe juntar con el resto de los integrantes de su grupo y los 6 juntos realizan las acciones que indican sus instrucciones. Cuando un grupo termina su juego le avisa a un Coordinador que le indica en qué orden término el grupo. Nota: maximizar la concurrencia; suponer que existe una función Jugar() que simula que los 6 integrantes de un grupo están jugando juntos; suponga que existe una función Romper(grupo) que simula cuando un jugador está rompiendo su cofre con el martillo y le retorna el grupo al que pertenece. **Respuesta:**

Proceso Jugador

```
task type Jugador is
end Jugador

Jugadores = array[0..29] of Jugador

task body Jugador is
begin
    Martillo.usar(miGrupo)
    Grupo[miGrupo].llegue(id)
    accept terminar(lugar : IN integer)
end
```

Proceso Martillo

```
task Martillo is
end Martillo

task body Martillo is
begin
    for i=1 to 30
    begin
        accept usar(miGrupo : OUT integer)
            Romper(miGrupo)
        end usar
    end for
end
```

Proceso Grupo

```
task type Grupo is
end Grupo

Grupos = array[0..4] of Grupo

task body Grupo is
begin
    for i=1 to 6
    begin
        accept llegue(idJugador)
            push(integrantes,idJugador)
        end llegue
    end for
    Jugar()
    coordinador.Fin(lugar)
    for i=1 to 6
    begin
        Jugador[pop(integrantes)].terminar(lugar)
    end
end
```

Proceso Coordinador

```
task type Coordinador is
end Coordinador

task body Coordinador is
begin
    for i=1 to 5
    begin
        accept Fin(lugar : OUT integer)
            lugar=i
        end Fin
    end
end
```

Consigna:
Resolver con PMS (Pasaje de Mensajes SINCRÓNICOS) el siguiente problema. Hay un teléfono público que debe ser usado por U usuarios de acuerdo al orden de llegada (se debe usar con exclusión mutua). El usuario debe esperar su turno, usa el teléfono y luego lo deja para que el siguiente lo use. Nota: cada usuario usa sólo una vez el teléfono. **Respuesta:**

Proceso Usuario

```
Process Usuario[1..U]{
    Buffer!pedir(id)
    Buffer?usar()
    //usa el telefono
    Buffer!sali()
}
```

Proceso Buffer

```
Process Buffer{
    for i=1 to U*2{
        do
        [] Usuario[*]?pedir(idUsuario) =>
            if libre{
                Usuario[idUsuario]!usar()
                libre=false
            }else{
                push(esperando,idUsuario)
            }
        [] Usuario[*]?sali() =>
            if empty(esperando){
                libre=True
            }else{
                Usuario[pop(esperando)]!usar
            }
        od
    }
}
```